

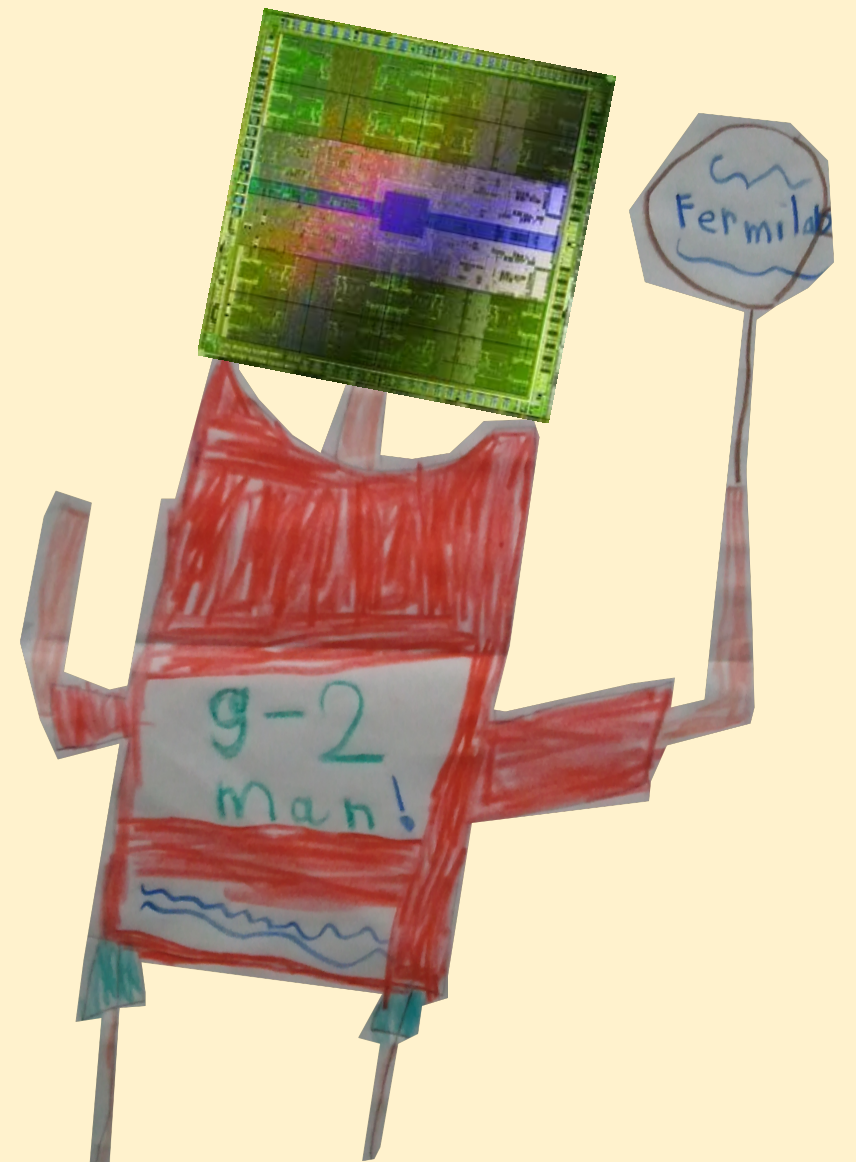
The Muon $g-2$ Computing “Big Picture”

Adam Lyon

(Fermilab/Scientific Computing Division)

$g-2$ for Beginners

February 20, 2013



I'll discuss the following topics:

What is the Fermilab computing vision and how do we fit in?

**Grids, Virtualization, Clouds
Art, Multicore, GPUs**

**I'll tell you how this matters to you
(yes, we all just want to make plots!)**

Many opinions here are mine alone

“Hopefully I'll tell you a little more than what you already know; certainly a little more than what I already know.” - R. Mohapatra



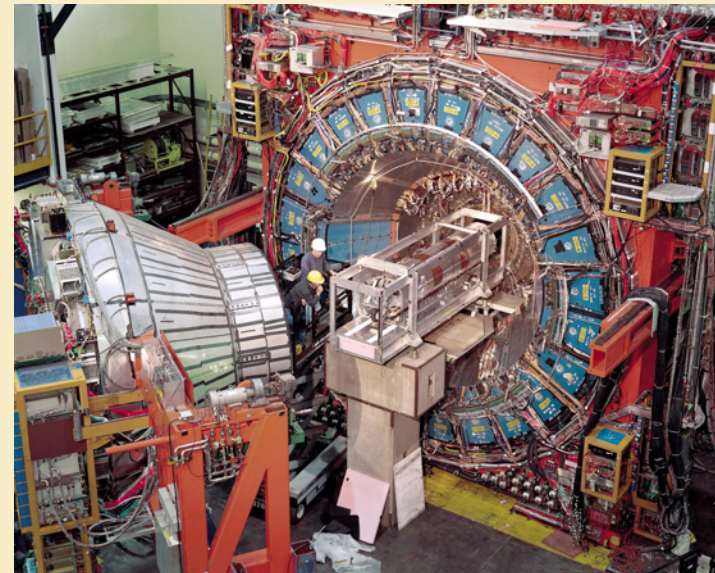
Enough topics here to interest you

My reading of the Lab's Computing Vision:

Efficiently use common computing resources and common tools to process data from the experiments.

It wasn't always this way –

**Run I & early Run II:
CDF & DØ had their own
separate and different computing farms with separate support
departments in the CD. Fixed target experiments had common
resources, but very minor compared to the collider experiments**



**In this era of doing more with less, we can no longer maintain
separate provincial systems. Migrate to common resources. IF
computing will largely complete this evolution.**

Fermilab provides many computing services

Fermilab has hosted many *batch* farms, but we now wrap farms in the *GRID*.

Grid computing:

A common interface to many batch systems on many farms. Common infrastructure and assistance



Open Science Grid

Main US DOE Grid

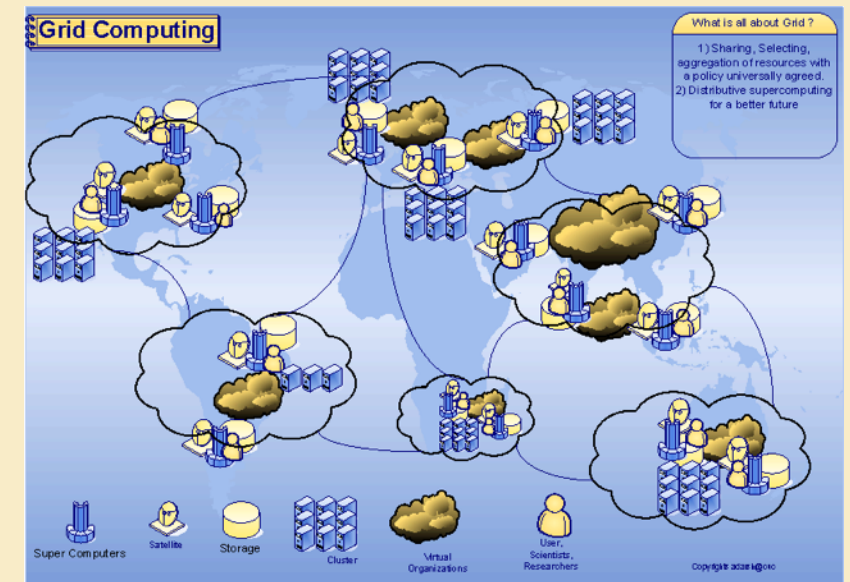
HEP, Biology, Seismology

**Not the same as NSF's
Supercomputer Xsede (Teragrid)**



What is a Grid?

A step in the direction of “computing as a utility”



- You submit your job, and it advertises requirements.**
- Sites advertise capabilities.**
- A “broker” matches your job to the site and it runs.**

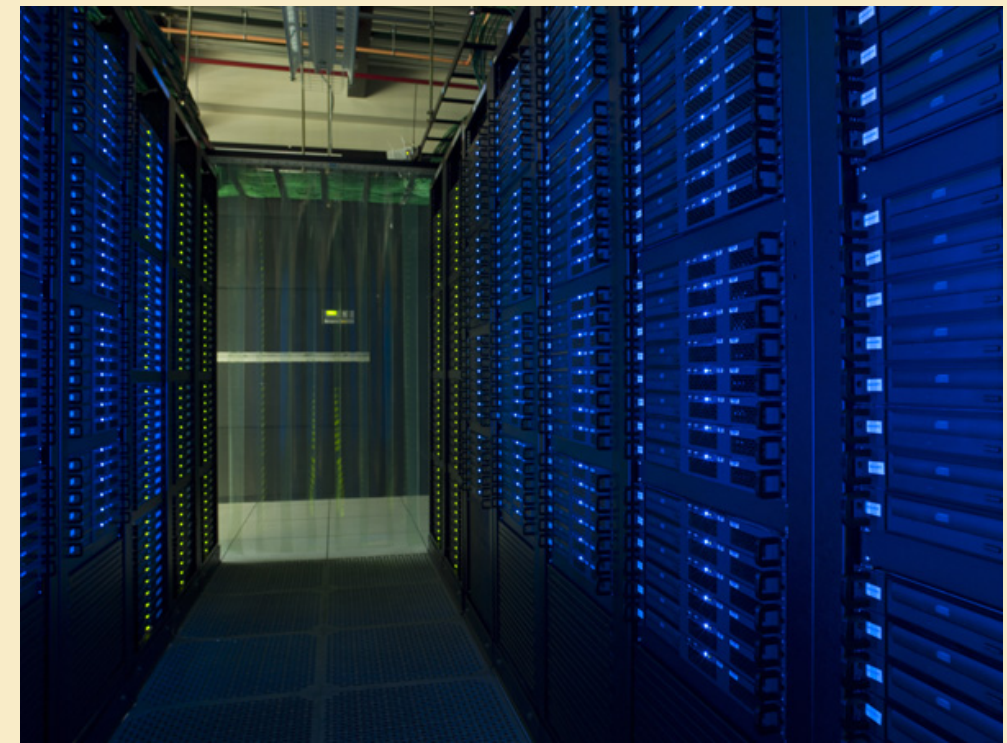
You don't care where your job runs, in principle

In practice it's been difficult to make this work smoothly

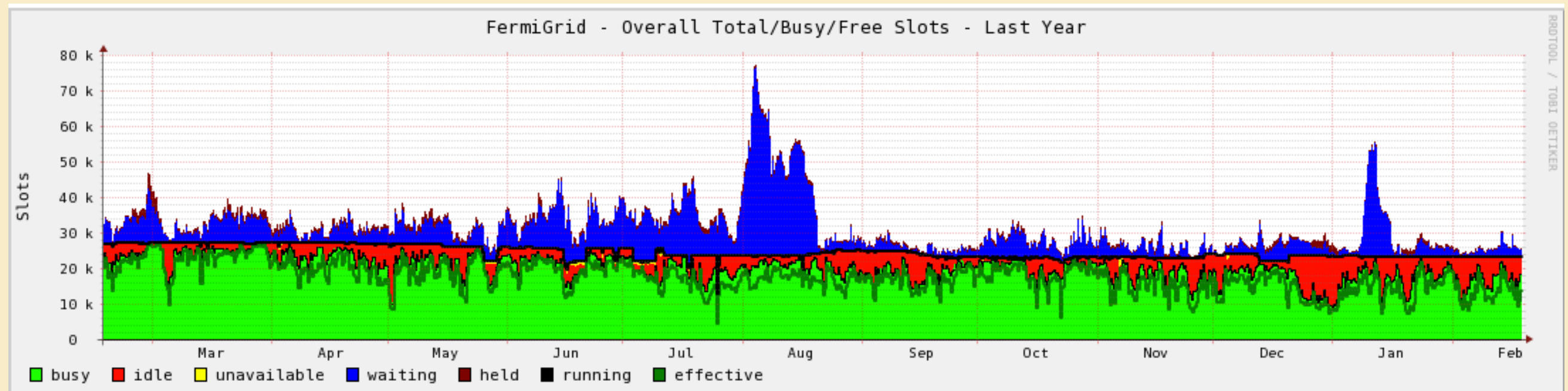
But the payoff? Opportunistic cycles! Even in the LHC era!

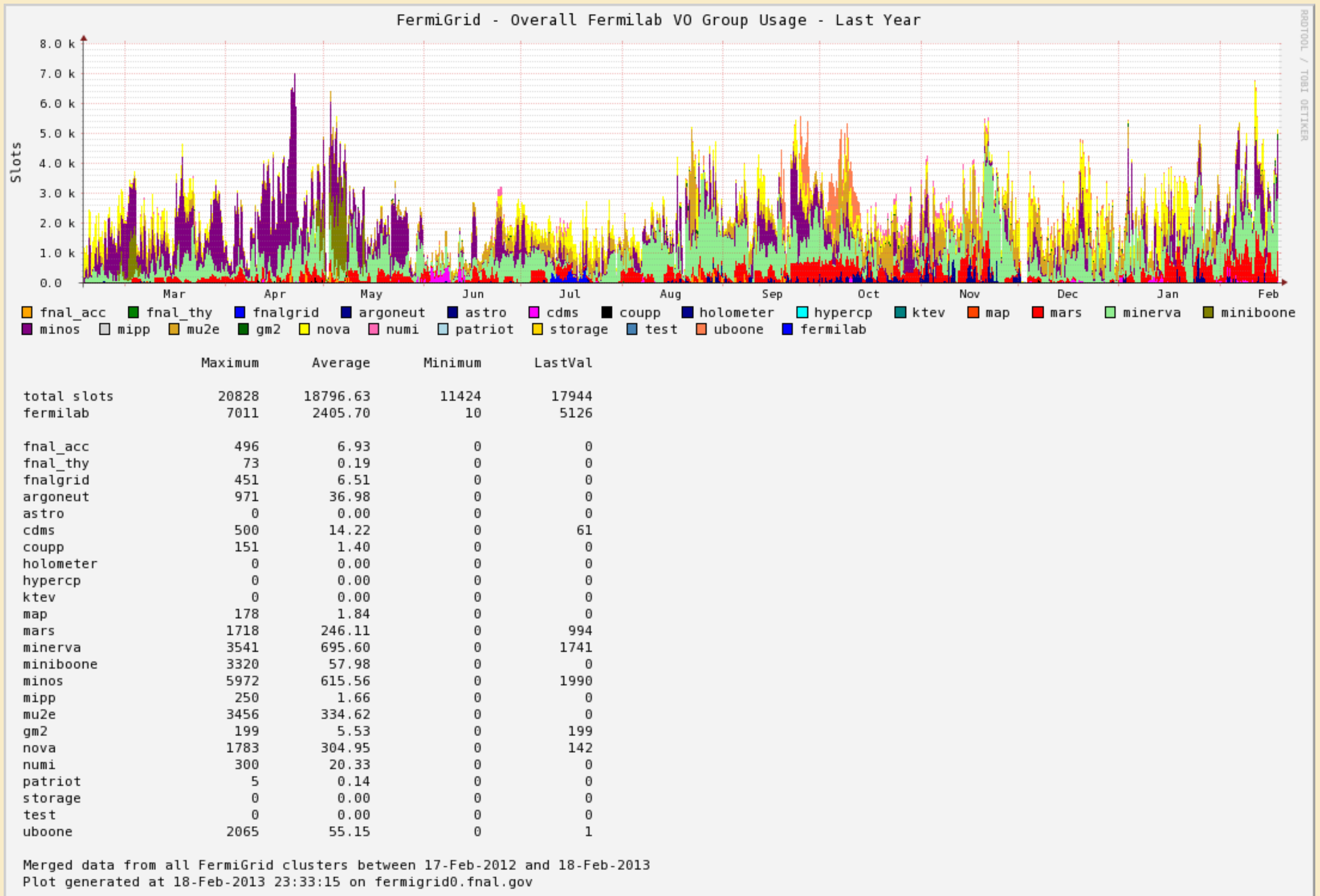
Our Fermilab Grid (Fermigrid)

**Fermigrid is a collection of farms at the lab all with an OSG interface:
CDF, DØ, CMS, General Purpose**



~25,000 slots! Separate farms cause some inefficiency





The Grid isn't easy!

Making different sites accept Grid requirements (better if they are LHC sites)

Authentication and authorization – are you who you say you are? Do you have permission to run there?

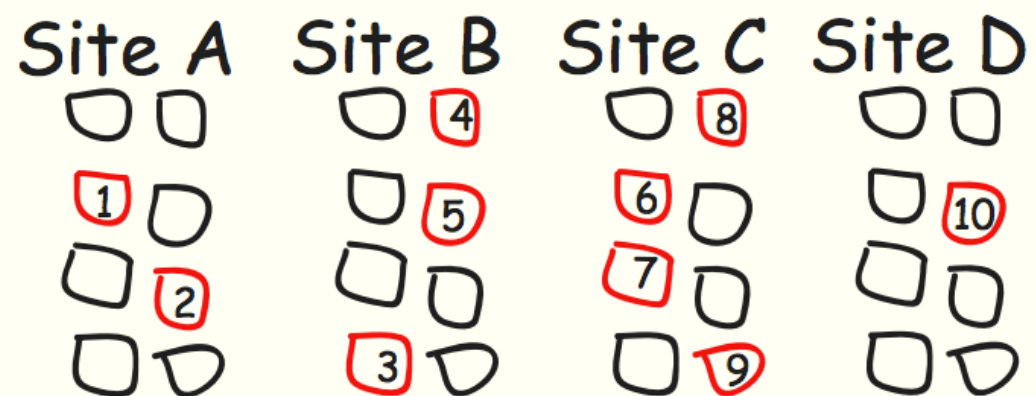
Do your job requirements really match the site?

How do you get your application and data to the job? How are the results returned?

Some solutions to these problems

“Glidein-WMS” – a batch system “projected” onto the Grid that looks local

The outside world sees:



 = worker node running pilot

We see



We submit to unified batch system we manage made up of worker nodes distributed around the grid and activated on demand

We can enforce our policies (fair share, time limits), but are subject to underlying grid system limits too

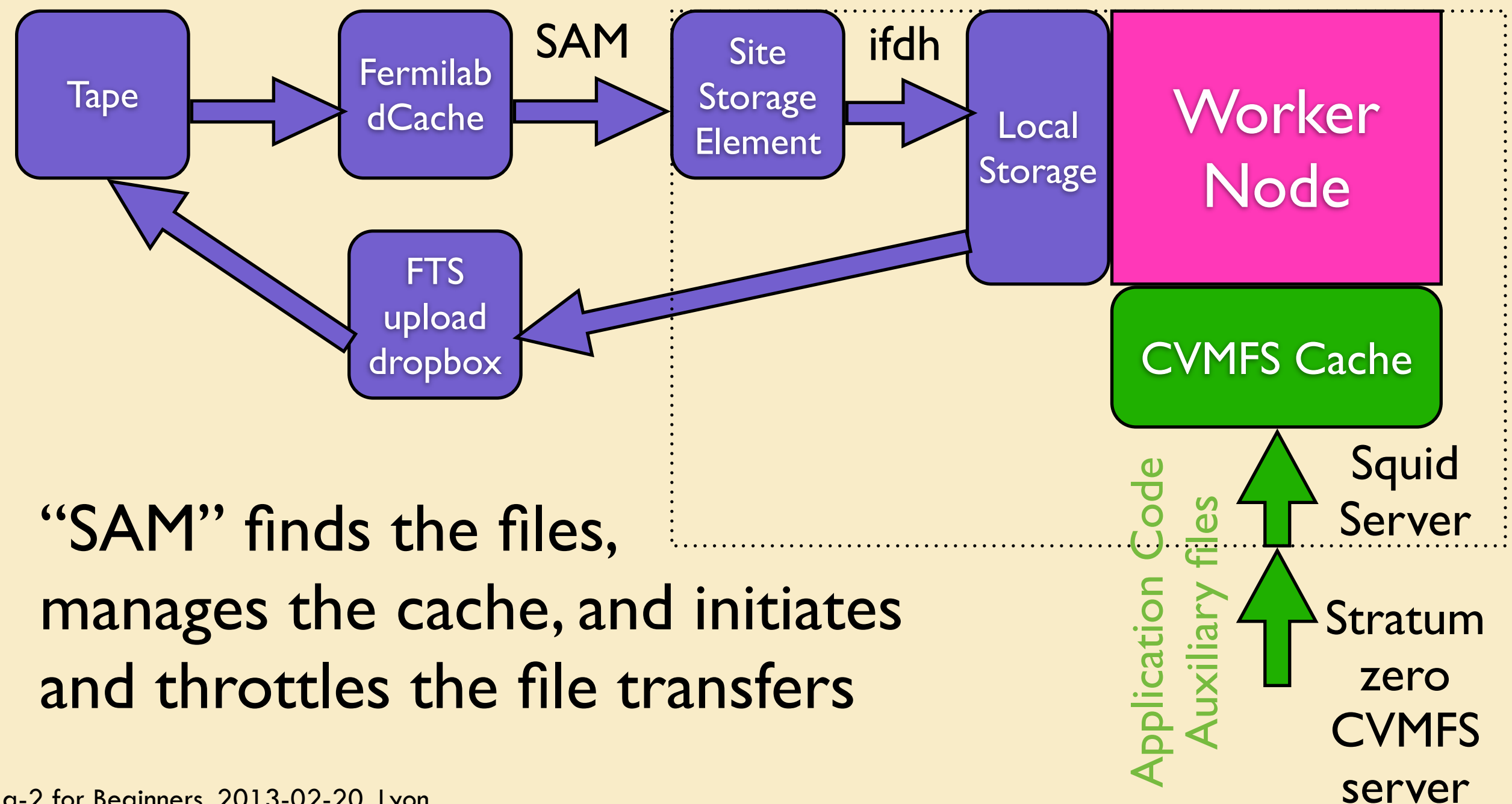
Pilots prevent jobs from landing on unsuitable worker nodes

Looks like one big batch system to us - easier to manage.

The “JobSub” package does this for you

Moving your data to your job

Note that the LHC experiments tried “move your job to the data” and are migrating to this method instead



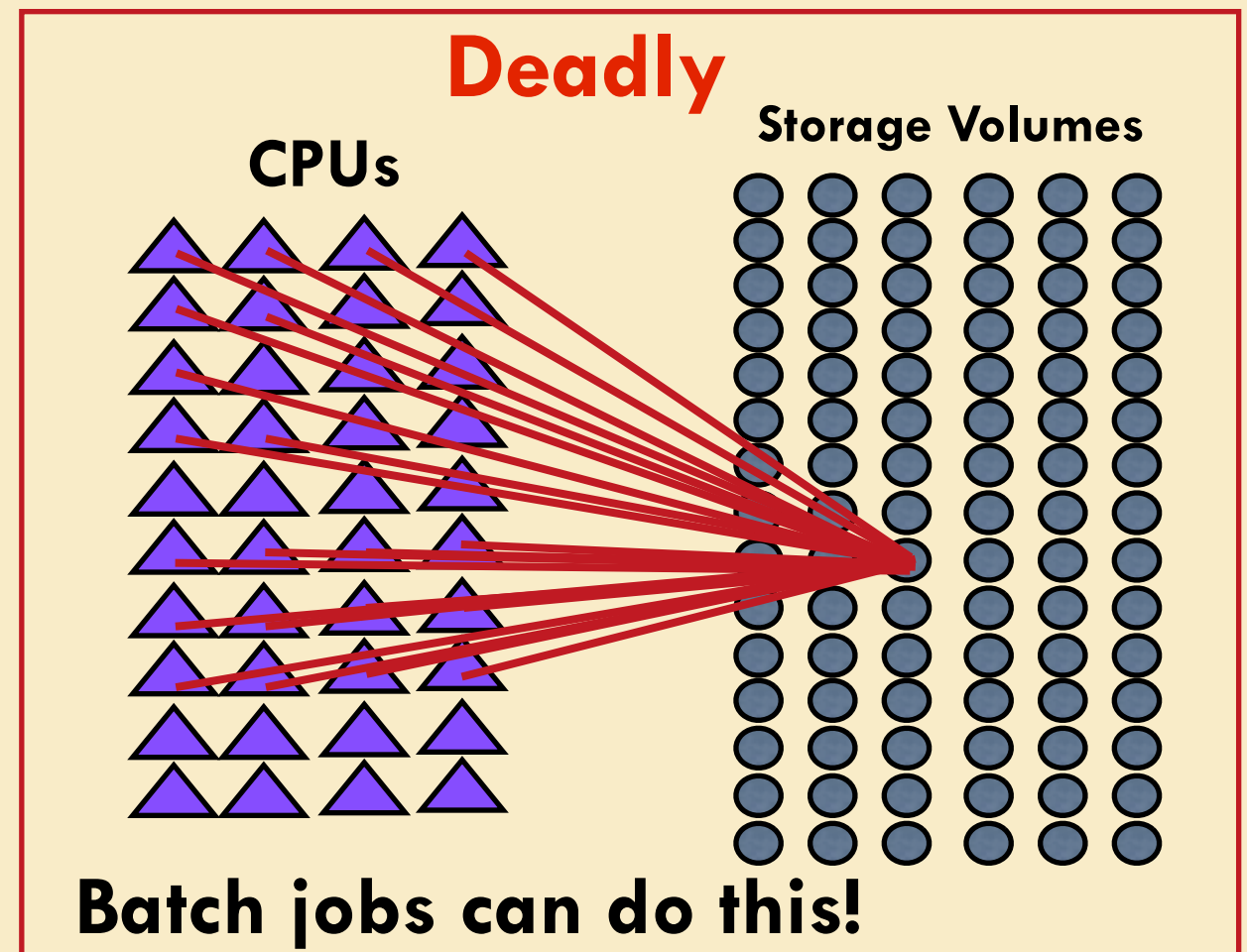
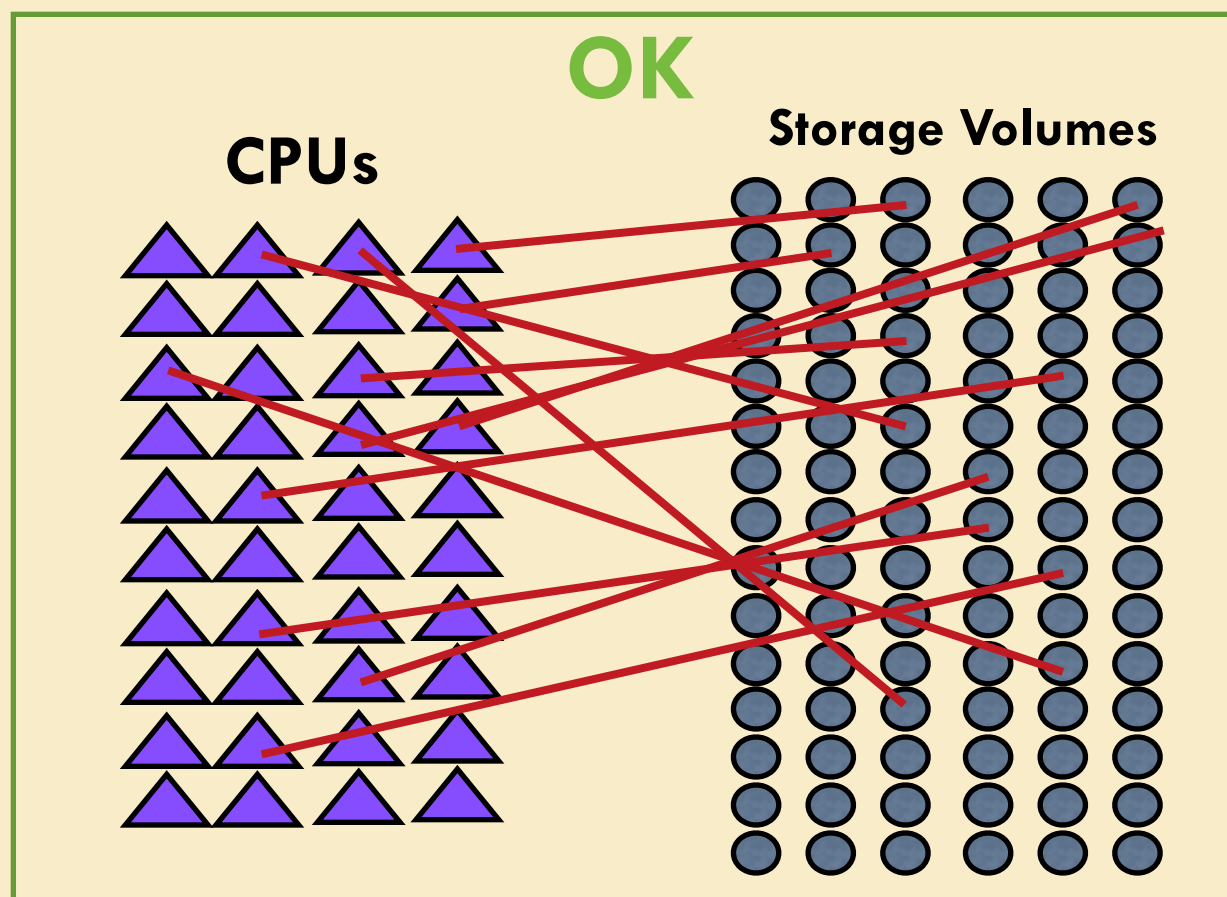
Local storage - Bluearc

Good news:

When used as designed, it works great

Bad news:

When used outside of its design, it kills computing for all of the IF experiments (hard to buy a robust, reasonably priced, 1 PB system)



Virtualization for efficiency



Problem: Most interactive and server machines are, on average, lightly used

Problem: “Bare metal” is expensive (to buy, power, & cool)

Solution: Virtualization

The bare metal runs several “virtual machines” (CPUs are optimized for this). Each virtual machine looks like a separate computer (separate IP address, own memory, etc). Typically one VM core per bare metal core, but can oversubscribe. A VM has ~5-10% overhead.

Our gm2gpvm0X machines are VMs



Cooling can be a hot problem



CERN computer room is “full”. High ceiling makes cooling difficult. Racks are only populated at 25% on average

The next step is “cloud” computing



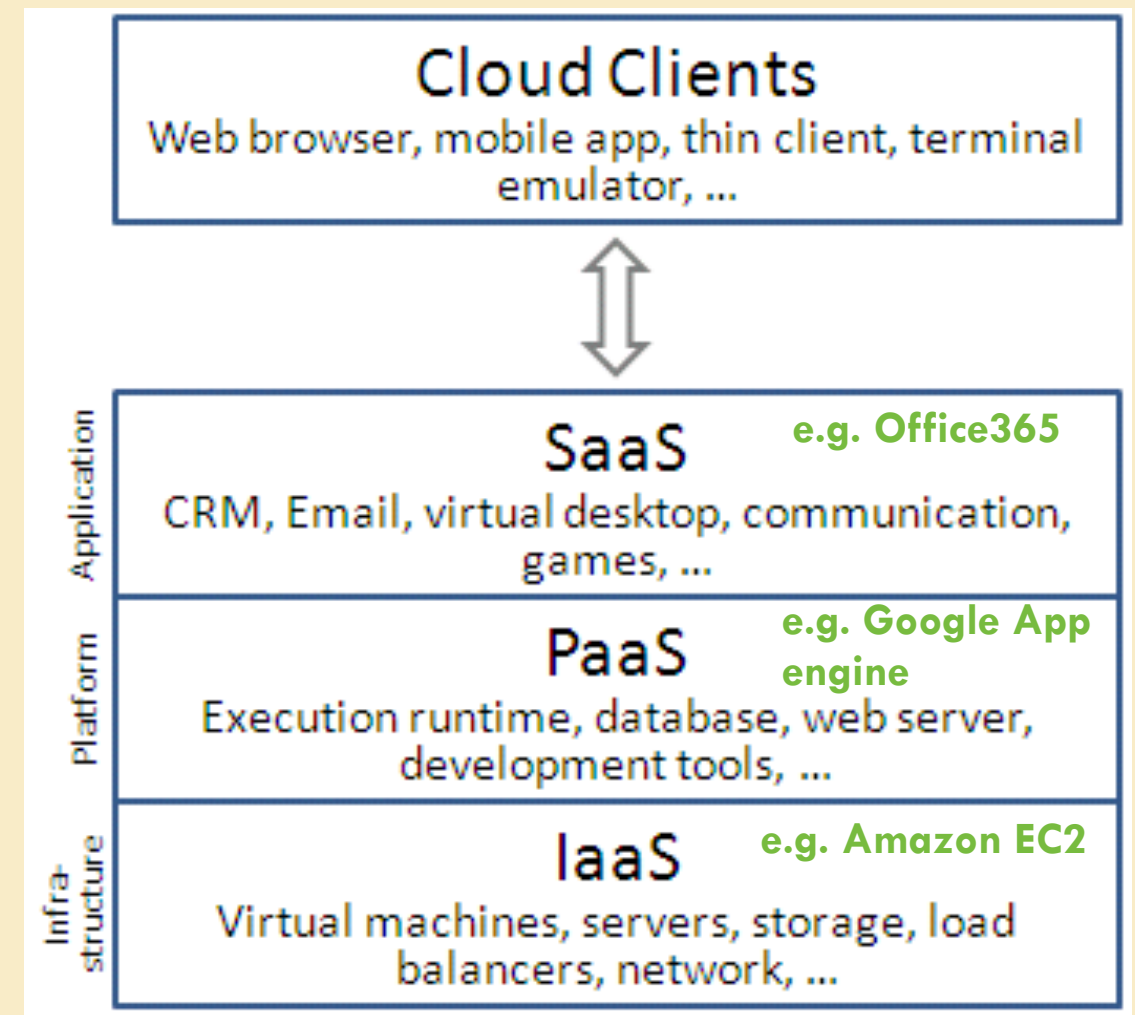
Cloud means to dynamically provision VMs for a specific purpose (e.g. you get your VM for you when you need it)

Maintenance is easier (can move VMs to different bare-metal for maintenance)

Efficient - “Fermicloud” is used instead of purchasing development machines. Some clouds can automatically add more VMs to handle increased load

“Cloud Bursting” can provide extra resources for a short period of time for a specific purpose - e.g. “Grid Bursting”

**Fermilab uses clouds for development and servers.
CERN will soon use their on cloud for nearly all computing (including business services)
[Amazon Elastic Cloud 2 - looking like a real utility - but expensive]**



Fermilab's place in all this

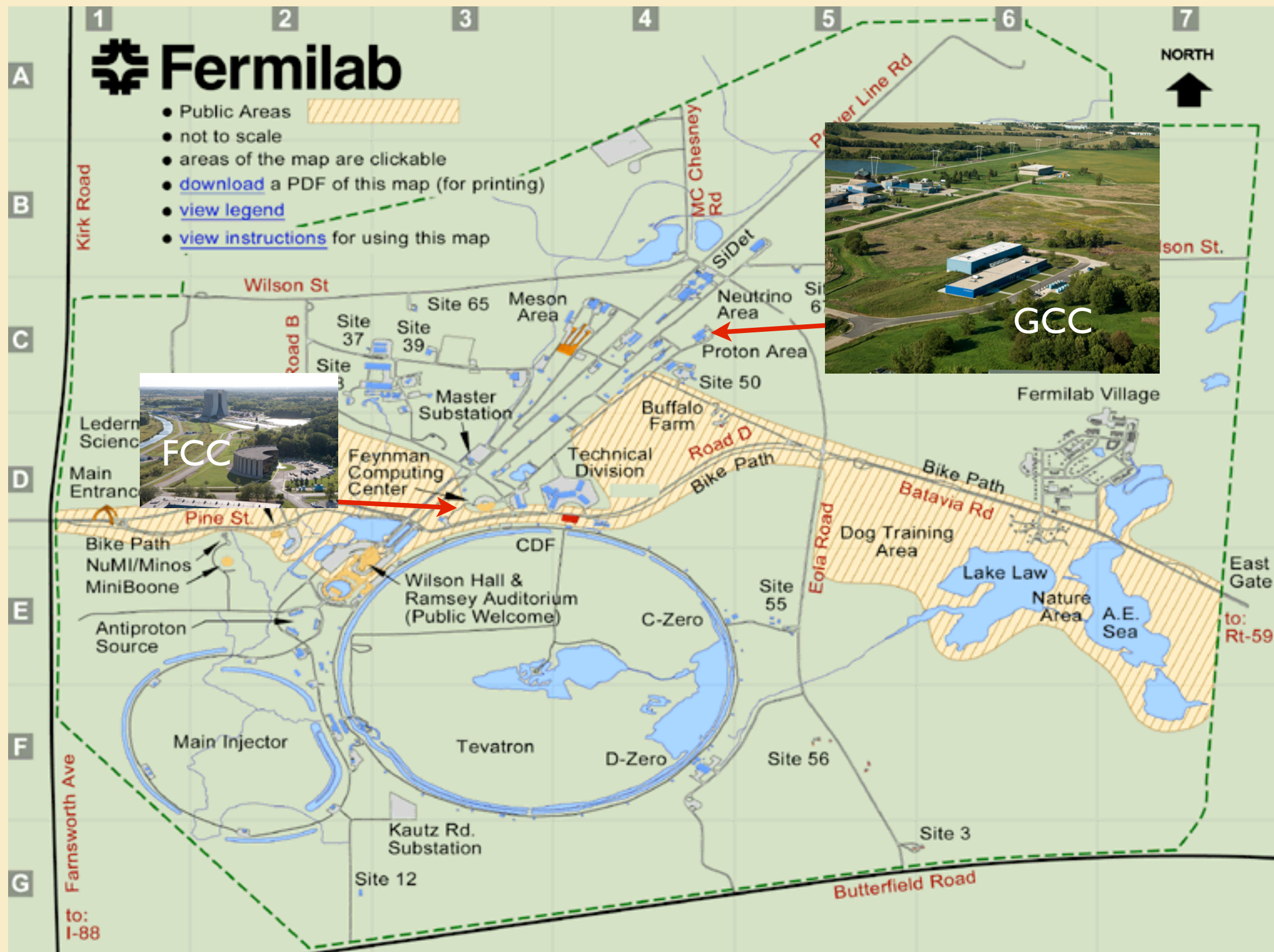
Fermilab is a leader in Grid computing [Both in implementation and development]. CERN and Argonne are leaders too. But Grid computing isn't as exciting as it used to be (it now kinda works)

Fermilab has embraced virtualization – using virtual machines for most everything now, except grid worker nodes

Fermilab's steps into the cloud are less coherent – three different cloud systems in use at the lab run by different groups. Ongoing R&D project to find which system fits our use cases the best. Recent review to come up with a lab strategy

I predict Fermilab will ultimately have a large cloud farm like CERN, because that's the easiest to manage with the fewest staff (and ultimately, that's the most important driver)

Fermilab's infrastructure



What's the status as per g-2?

For the intensity frontier experiments:

JobSub submits your jobs to Fermigrid

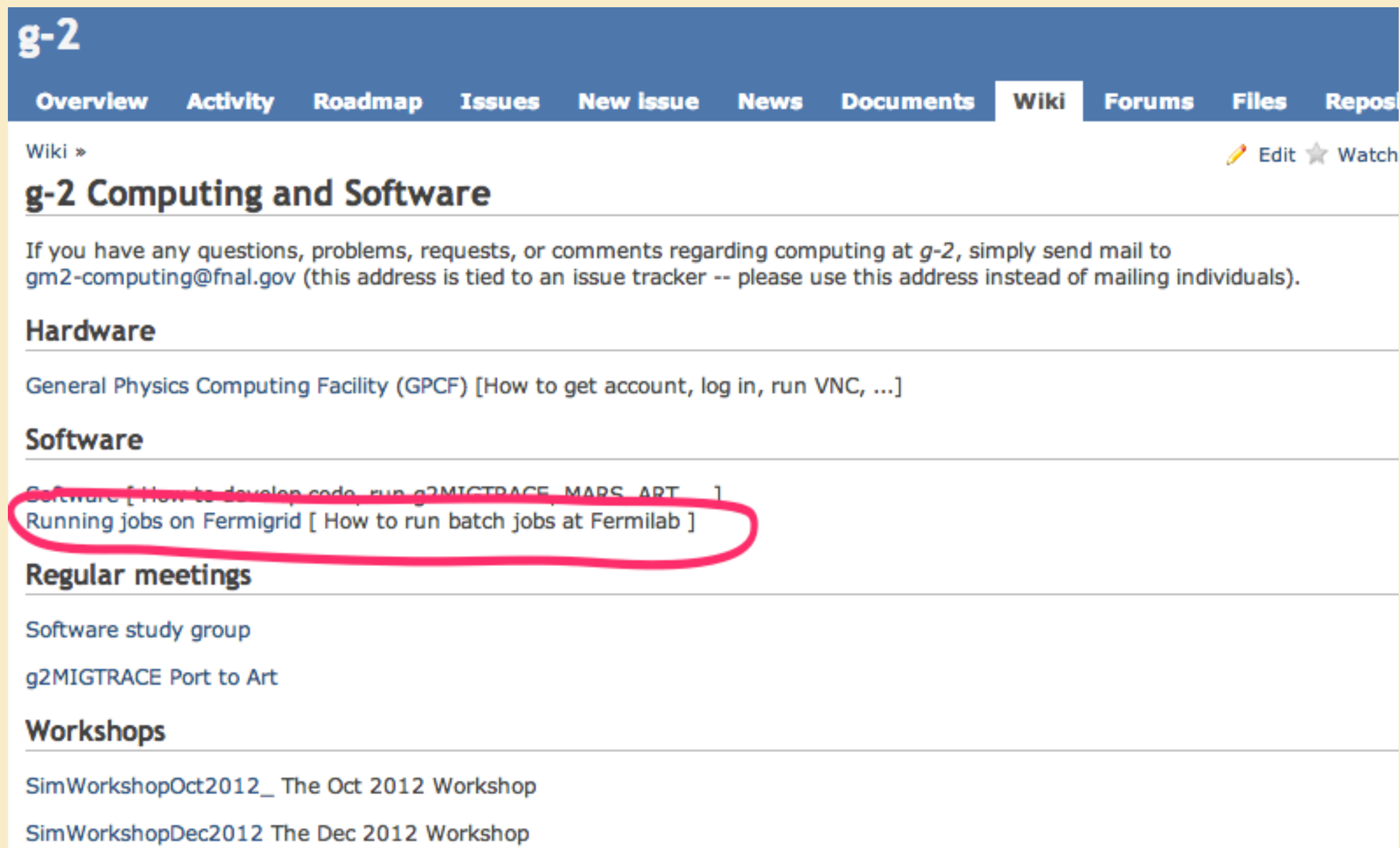
Working to add submission to remote sites

CVMFS is being tested, some decisions need to be made (will Fermilab trust a remote main server?). Expect deployment in the spring. This service will have a major impact on making the Grid easier to use. OSG is making a service too. We're using it for OSX distribution

SAM is being improved for IF. Making simplifications and improving the user interface (now web based). We have a SAM instance and will be getting cache soon

What do I do about this (I just wanna make plots)?

You don't need to do much... Carefully read and understand...



The screenshot shows the 'g-2' Wiki page. At the top is a blue navigation bar with tabs: Overview, Activity, Roadmap, Issues, New Issue, News, Documents, Wiki (selected), Forums, Files, and Repos. Below the navigation bar, the page title is 'g-2 Computing and Software'. A paragraph of text follows: 'If you have any questions, problems, requests, or comments regarding computing at g-2, simply send mail to gm2-computing@fnal.gov (this address is tied to an issue tracker -- please use this address instead of mailing individuals)'. Below this are several sections: 'Hardware' with a link to 'General Physics Computing Facility (GPCF) [How to get account, log in, run VNC, ...]'; 'Software' with two links: 'Software [How to develop code, run g2MIGTRACE, MARS, ART ...]' and 'Running jobs on Fermigrid [How to run batch jobs at Fermilab]', which is circled in red; 'Regular meetings' with links to 'Software study group' and 'g2MIGTRACE Port to Art'; and 'Workshops' with links to 'SimWorkshopOct2012_ The Oct 2012 Workshop' and 'SimWorkshopDec2012 The Dec 2012 Workshop'.

g-2

Overview Activity Roadmap Issues New Issue News Documents **Wiki** Forums Files Repos

Wiki » [Edit](#) [Watch](#)

g-2 Computing and Software

If you have any questions, problems, requests, or comments regarding computing at g-2, simply send mail to gm2-computing@fnal.gov (this address is tied to an issue tracker -- please use this address instead of mailing individuals).

Hardware

[General Physics Computing Facility \(GPCF\) \[How to get account, log in, run VNC, ...\]](#)

Software

[Software \[How to develop code, run g2MIGTRACE, MARS, ART ... \]](#)

[Running jobs on Fermigrid \[How to run batch jobs at Fermilab \]](#)

Regular meetings

[Software study group](#)

[g2MIGTRACE Port to Art](#)

Workshops

[SimWorkshopOct2012_ The Oct 2012 Workshop](#)

[SimWorkshopDec2012 The Dec 2012 Workshop](#)

What do I do about this (I just wanna make plots)?

How to set things up (e.g. get certificates, etc)

How to write a script to run your program

How to use IFDH to protect the Bluearc

How to move files to the worker node

How to copy output files back to /gm2/data

How to submit your job using JobSub

Common Tools - Art

Writing a Framework for multiple experiments - a new thing!

Tevatron Run I Software: Fortran, Zebra banks, CERNLIB, PAW

After Tevatron Run I:

Fortran .EQ. 'ick'

C++ == "great! let's convert everything to C++ and OO"

Was moving a good idea? Perhaps now it's looking better. 'Root' was a big driver. Created the new position of C++ expert (we still need them). Some people had more fun writing C++ to do physics than doing the physics

We had to keep up with the state-of-the-art

Why we need C++ experts

Non-trivial object structure is hard to get right (look at some of the ugly Root classes) and hard to fix when entrenched

Minimizing dependencies in large systems is very difficult

Memory management can be deadly (leaks)

Some very advanced techniques are useful, but difficult (templates, generic programming)

Build inconsistencies are very difficult to catch

We have three C++ experts in the SCD

Motivation for frameworks

You concentrate on the physics code, let the framework handle...

Data input/output (e.g. complicated interactions with data handling)

The “event store” (some catalog of event data that you access)

Looping over events

Access to services

Plugin Modularity

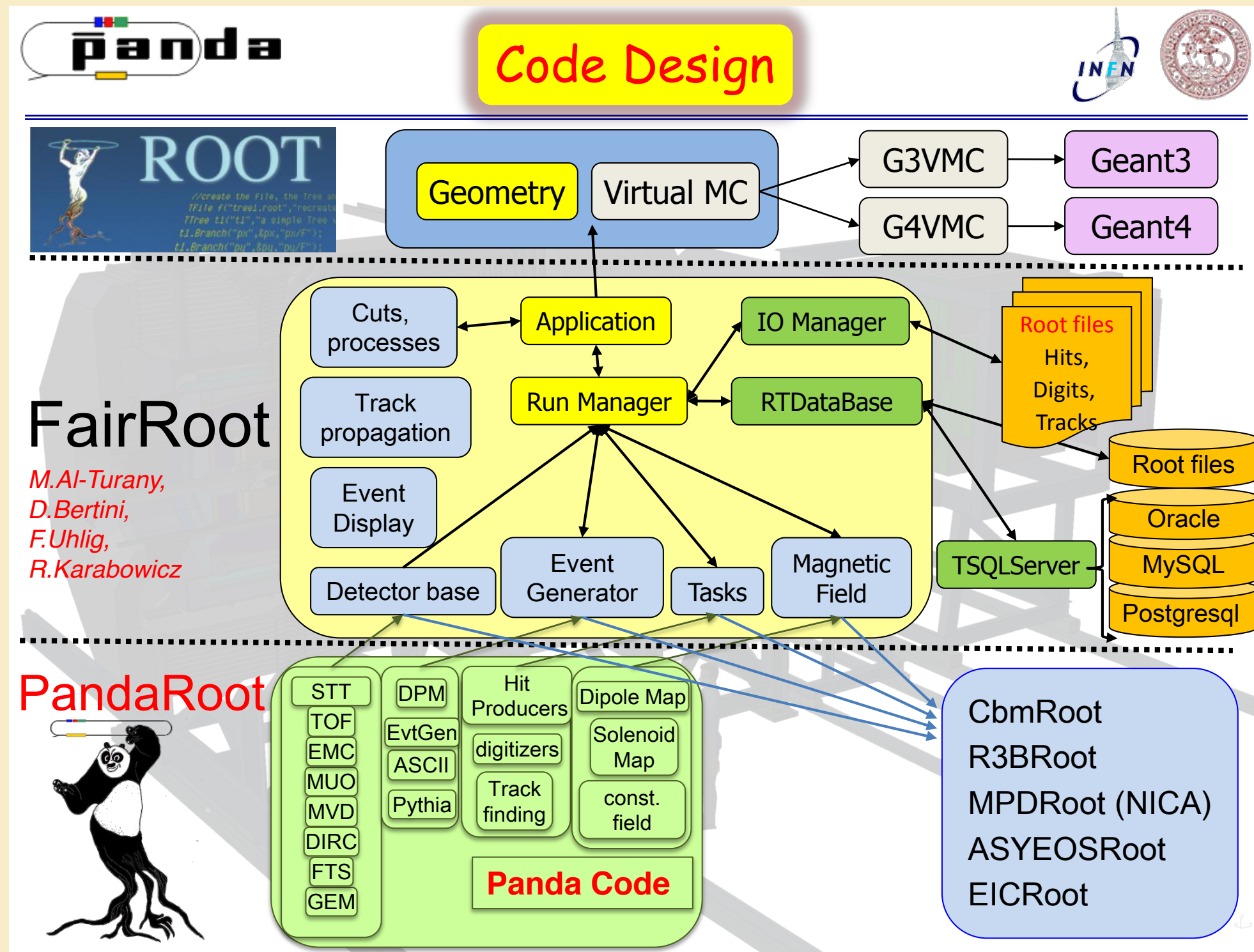
Make it easier to work together

Prior to ART, nearly every experiment wrote their own framework system (sometimes several). Very little sharing of ideas and innovations

Writing a framework is fun (for some people) ... at first ... then the details get difficult

Root is a common theme for Frameworks

Another framework example



Art History

SCD CET Group (Emergent technologies) contributed heavily to the CMS framework

Mu2e didn't want to write their own Framework – wanted to use the CMS framework without the “CMSness”

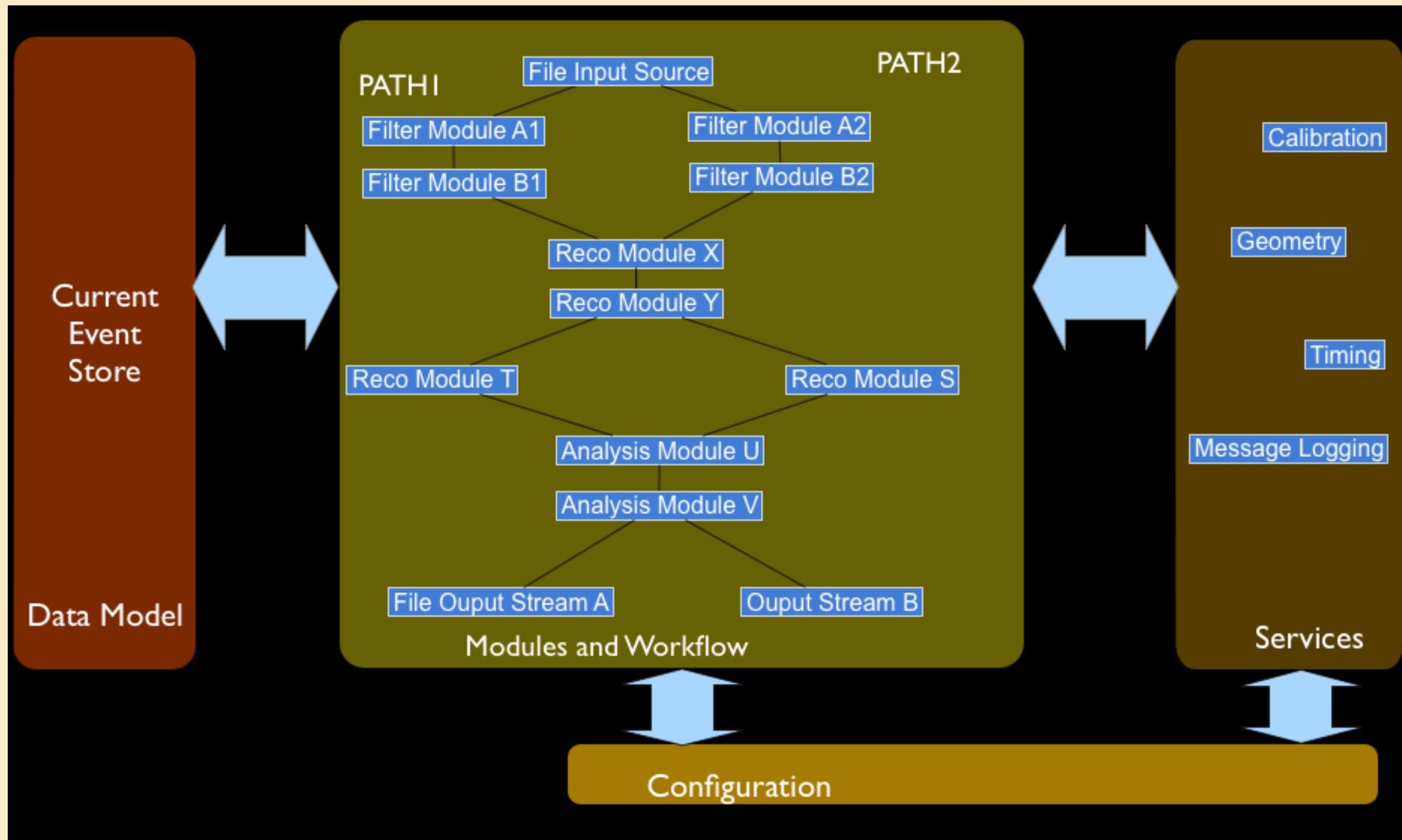
CET adapted CMS framework into “CMS-Lite”

Cleaned up the CMS code, reversed some controversial design decisions, diverged -- renamed Art (no particular meaning)

Adopted by Mu2e, NOvA, LArSoft [Microboone, LBNE], DS50, g-2

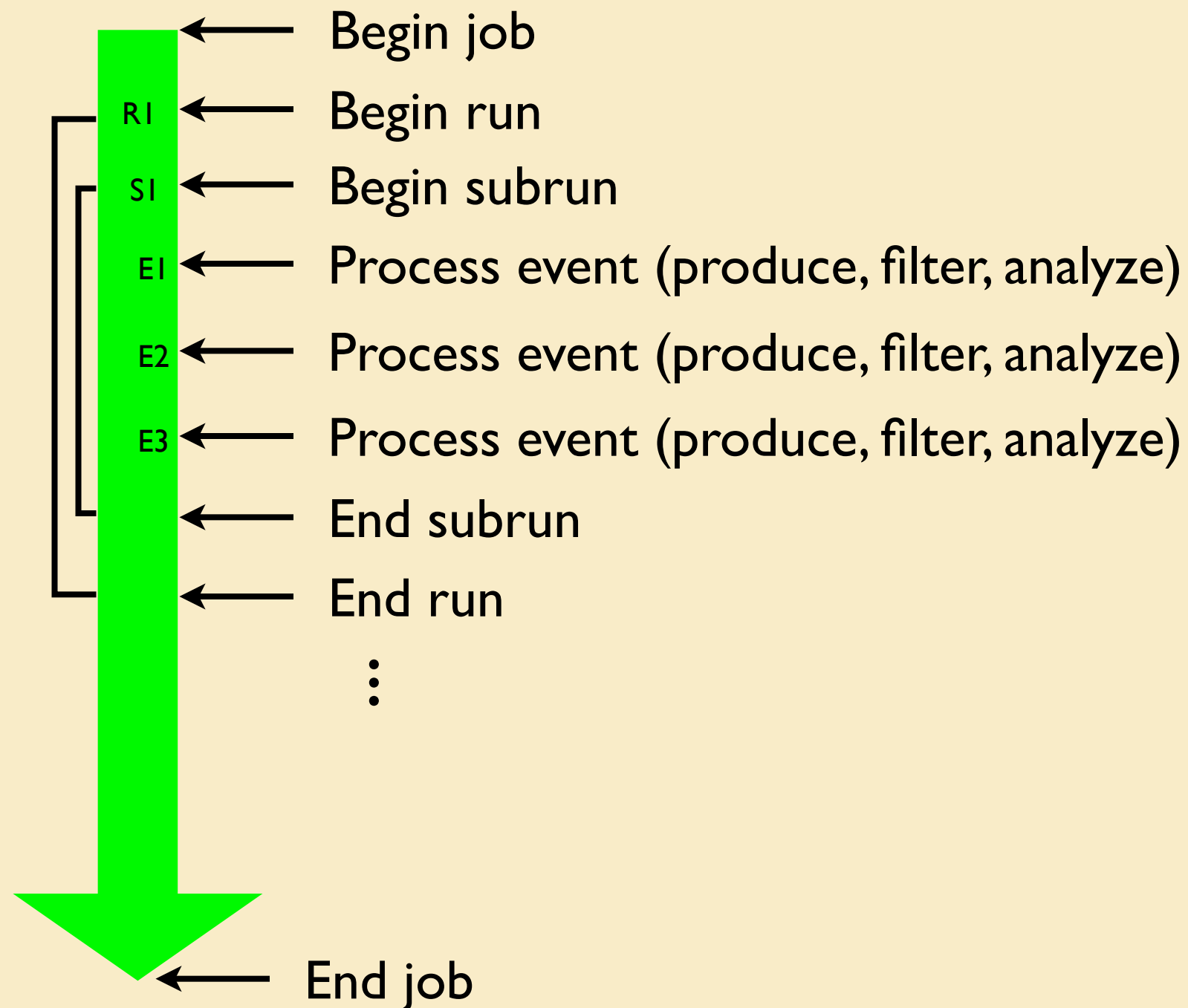
Probably best use cases is NOvA (real data) - very successful

Art's 4 pieces



What do you write?

You write modules that can access data and do things at certain times



What do you write

You can also write “services” for globally accessible information (e.g. geometry information, calibration constants)

Services can also have hooks too

Globally accessible objects (e.g. singletons) are not trivial to write in C++. Services makes them easy. You just write an object

To access (quite simple):

```
art::ServiceHandle< gm2::GeometryDB > geomDB;  
geomDB->getWorldVolume();
```


Implications

Allows you to write your physics code without worrying about the infrastructure. Makes it easy to work with others. But not for free – you have to learn it.

Some people find such a system constraining:

Infrastructure is hidden behind the scenes from you

Your ideas may not be included

You have to trust a system you didn't write

You miss out on the fun of writing super-cool complicated C++ code

Some people find such a system liberating:

You can concentrate on physics code

Your C++ is pretty easy (you are *using* a complicated system, not *writing* it)

You get to miss out having to maintain the complicated system (yay!)

You can use code from others and share yours with others

You can get services for free (e.g. data handling)

In g2migtrace/src/primaryConstruction.cc

```
// constructionMaterials is essentially a "materials library" class.
// Passing to to construction functions allows access to all materials

/**** BEGIN CONSTRUCTION PROCESS ****/

// Construct the world volume
labPTR = lab -> ConstructLab();
// Construct the "holders" of the actual physical objects
#ifdef TESTBEAM
    Arch.push_back(labPTR);
#else
    Arch = arc->ConstructArcs(labPTR);
#endif
// Build the calorimeters
// cal -> ConstructCalorimeters(Arch);
// station->ConstructStations(Arch);
#ifdef TESTBEAM
// Build the physical vacuum chambers and the vacuum itself
Vach = vC -> ConstructVacChamber(Arch);
```

In g2migtrace/src/primaryConstruction.cc

```
// constructionMaterials is essentially a "materials library" class.  
// Passing to to construction functions allows access to all materials  
  
/**** BEGIN CONSTRUCTION PROCESS ****/  
  
// Construct the world volume  
labPTR = lab -> ConstructLab();  
// Construct the "holders" of the actual physical objects  
#ifdef TESTBEAM  
    Arch.push_back(labPTR);  
#else  
    Arch = arc->ConstructArcs(labPTR);  
#endif  
// Build the calorimeters  
// cal -> ConstructCalorimeters(Arch);  
    station->ConstructStations(Arch);  
#ifndef TESTBEAM  
    // Build the physical vacuum chambers and the vacuum itself  
    Vach = vC -> ConstructVacChamber(Arch);
```

We can't maintain code like this and our sanity

In g2migtrace/src/primaryConstruction.cc

```
// constructionMaterials is essentially a "materials library" class.  
// Passing to to construction functions allows access to all materials
```

```
/*  
**** BEGIN CONSTRUCTION PROCESS ****  
*/
```

WHAT IF WE WANT TO
TEST A DIFFERENT
DETECTOR?

```
// Construct the world volume
```

```
labPTR = lab -> ConstructLab();
```

```
// Construct the "holders" of the actual physical objects
```

```
#ifdef TESTBEAM
```

```
Arch.push_back(labPTR);
```

```
#else
```

```
Arch = arc->ConstructArcs(labPTR);
```

```
#endif
```

THIS CODE CRASHES IN OPTIMIZED
BUILDS. #IFDEF CREATED A
SUBTLE HARD TO FIND BUG

```
// Build the calorimeters
```

```
// cal -> ConstructCalorimeters(Arch);
```

```
station->ConstructStations(Arch);
```

```
#ifndef TESTBEAM
```

```
// Build the physical vacuum chambers and the vacuum itself
```

```
Vach = vC -> ConstructVacChamber(Arch);
```

THIS KIND OF CODE IS
VERY HARD TO EXCISE
LATER

We can't maintain code like this and our sanity

A configuration system is crucial

```
#include "geom/world.fcl"
#include "geom/arc.fcl"
#include "geom/vac.fcl"
#include "geom/inflector.fcl"
#include "geom/quad.fcl"
#include "geom/oct.fcl"
#include "geom/kicker.fcl"
#include "geom/fiberHarp.fcl"
#include "geom/station.fcl"
#include "geom/calorimeter.fcl"
#include "geom/collimator.fcl"
#include "geom/PGA.fcl"
#include "geom/g2GPS.fcl"
#include "geom/traceback.fcl"

process_name:myProcessName

source: {
  module_type: EmptyEvent
  maxEvents: 10
}

services: {
  message : {
    debugModules : ["*"]
    suppressInfo : []

    destinations : {
      LogToConsole : {
        type : "cout"
        threshold : "DEBUG"
      }
      LogToFile : {
        type : "file"
        filename : "gm2ringsim.log"
        append : false
        threshold : "DEBUG"
      }
    }
  }
}
```

```
user : {

  // Mandatory ArtG4 services
  DetectorHolder: {}
  ActionHolder: {}
  PhysicsListHolder: {}
  RandomNumberGenerator: {}

  // Geometry
  Geometry: {
    world: @local::world_geom
    arc: @local::arc_geom
    vac: @local::vac_geom
    inflector: @local::inflector_geom
    quad: @local::quad_geom
    octupole: @local::oct_geom
    kicker: @local::kicker_geom
    fiberHarp: @local::fiberHarp_geom
    station: @local::station_geom
    calorimeter: @local::calorimeter_geom
    collimator: @local::collimator_geom
    pga: @local::PGA_geom
    traceback: @local::traceback_geom
  }

  // Global simulation settings
  RunSettings : {
    SpinTracking : {
      spinTrackingEnabled : false
    }
    G2GPSSettings: @local::G2GPS_downstreamInflectorMandrel
  }

  // Action(s) for the simulation
  StackingAction: {
    name: "stackingAction"
    minWavelength:250 //nm
    maxWavelength:950 //nm
  }

  physicalVolumeStore: {}
}
```

Configuration system

```
Gm2PhysicsList: {}

TrackingAction : {
    name : "trackingAction"
}

ClockAction: {}

G2PGA: {
    name: "primary"
}

// Detectors
World: {}
Arc: {}
VacuumChamber: {}
Inflector: {}
Quad: {}
Octupole: {}
Kicker: {}
FiberHarp: {}
Station: {}
Calorimeter: {}
Collimator : {}
Traceback: {}

} //user:
} //services:

outputs: {
  out1: {
    module_type: RootOutput
    fileName: "gm2ringsimout.root"
  }
}

physics: {
  producers: {
    artg4 : {
      module_type: artg4Main
      enableVisualization: false
      macroPath: ".../macros:../macros:../..//macros"
```

```
    visMacro: "vis.mac"
    afterEvent: pause // (ui, pause, pass)
  }

  analyzers: {
    readTracks: {
      module_type: particleTrackAnalyzer
    }
  }

  path1: [ artg4 ]
  readPath: [ readTracks ]
  stream1: [ out1 ]

  trigger_paths: [ path1 ]
  end_paths: [ readPath, stream1 ]
}
```

**Using a framework
allows for “free lunch”
improvements (e.g. Art integration
with SAM data handling) and ...**

Multiprocessor/Multicore

Nearly all modern machines are multi-cpu

Multiprocessor: more than one CPU chip

Multicore: more than one CPU on the same chip

Newest Fermigrid worker nodes have 64 cores (!!)

In a batch system, using these cores is easy:

1 job (slot) per core (actually, we oversubscribe a little)

HEP processing lends itself to “trivial” parallel programming

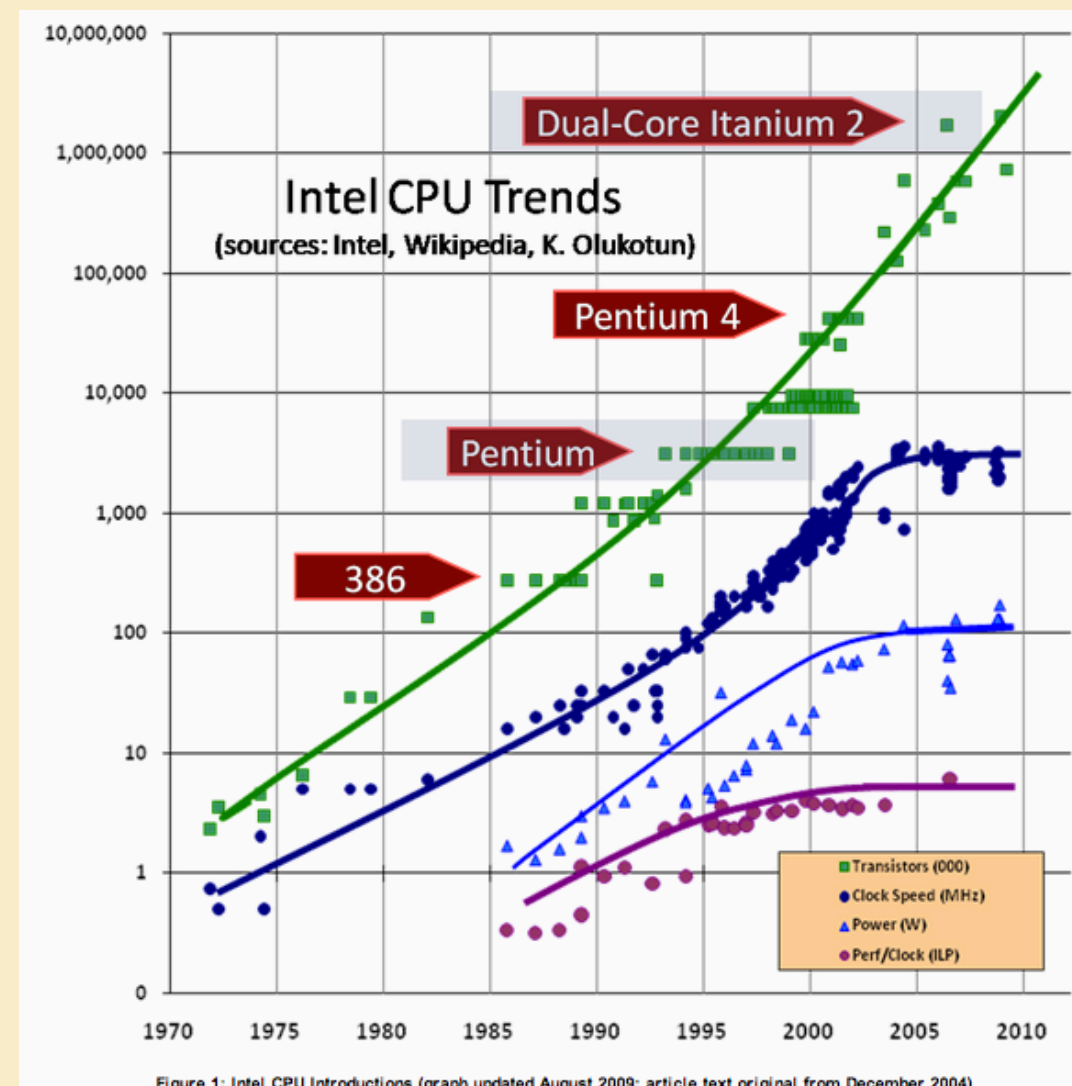
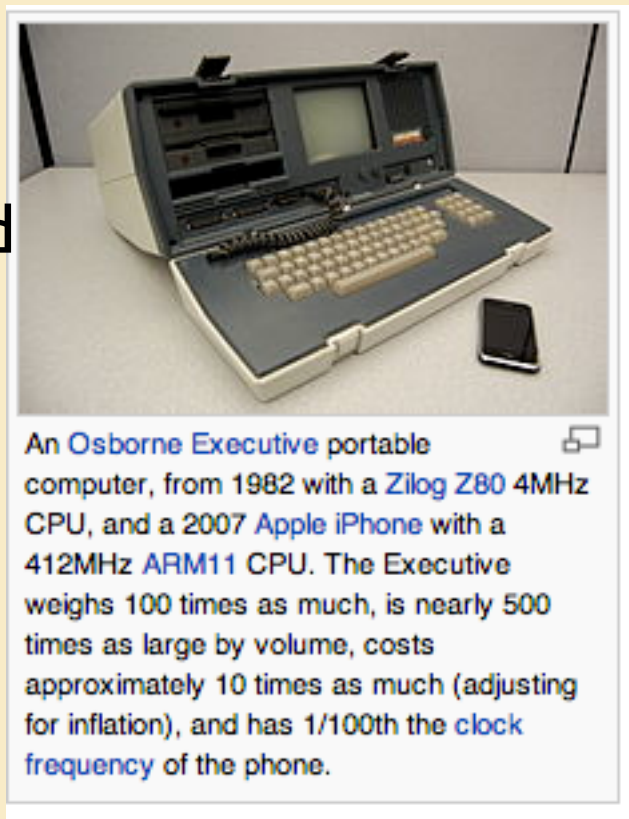
Each event is processed by one core in isolation

Note that this is very different from “super-computing” (or HPC) where the parallel processes share information

The Free Lunch is Over

Historically, CPU speed doubled every 18 months - but no longer

My awesome
computer I had
when an
undergrad

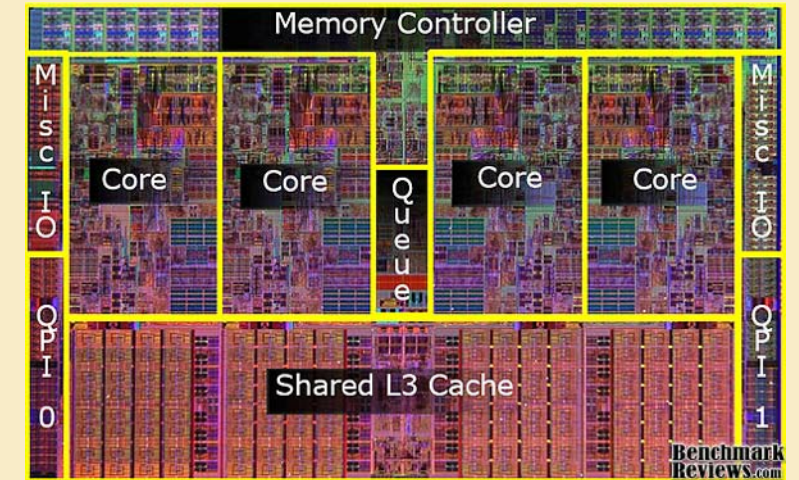


Why don't we have 10 GHz CPUs? Heat dissipation, power consumption, current leakage (see <http://www.gotw.ca/publications/concurrency-ddj.htm>)

The increase in speed had been useful, sometimes crucial, for experiments

More cores is not necessarily better

**Instead of speed doubling in 18 months,
Number of cores doubles
(Can't make them faster, so give you more of them)**



**But more cores means you need more memory
Fortunately, memory prices halve every ~18 months**

The standard seems to be 2 GB/core - that's what we buy

Many experiments (e.g. CMS, ATLAS) cannot fit their reconstruction processing of an event in 2 GB, especially for large pile up!!

**One solution is to not use all of the cores on the machine
(e.g. instead of using 16 cores/machine, use 8) so get > 2 GB/core**

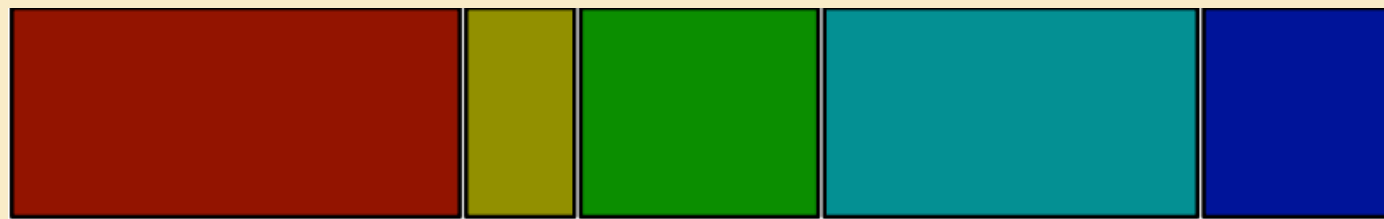
But that's wasteful and difficult to manage

New paradigm - multicores per event

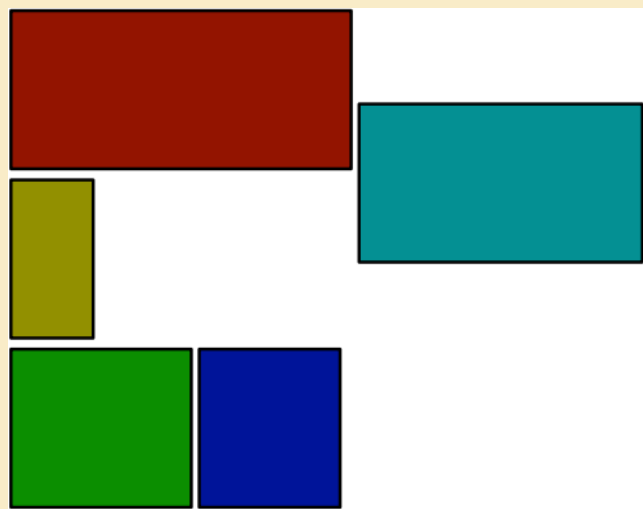
Split up the processing of an event among multiple cores (concurrency or threading).

HEP processing is task based (e.g. cluster hits, find tracks, fit tracks, reconstruct e's)

Traditional



Parallel



Difficulties: Handling task dependencies

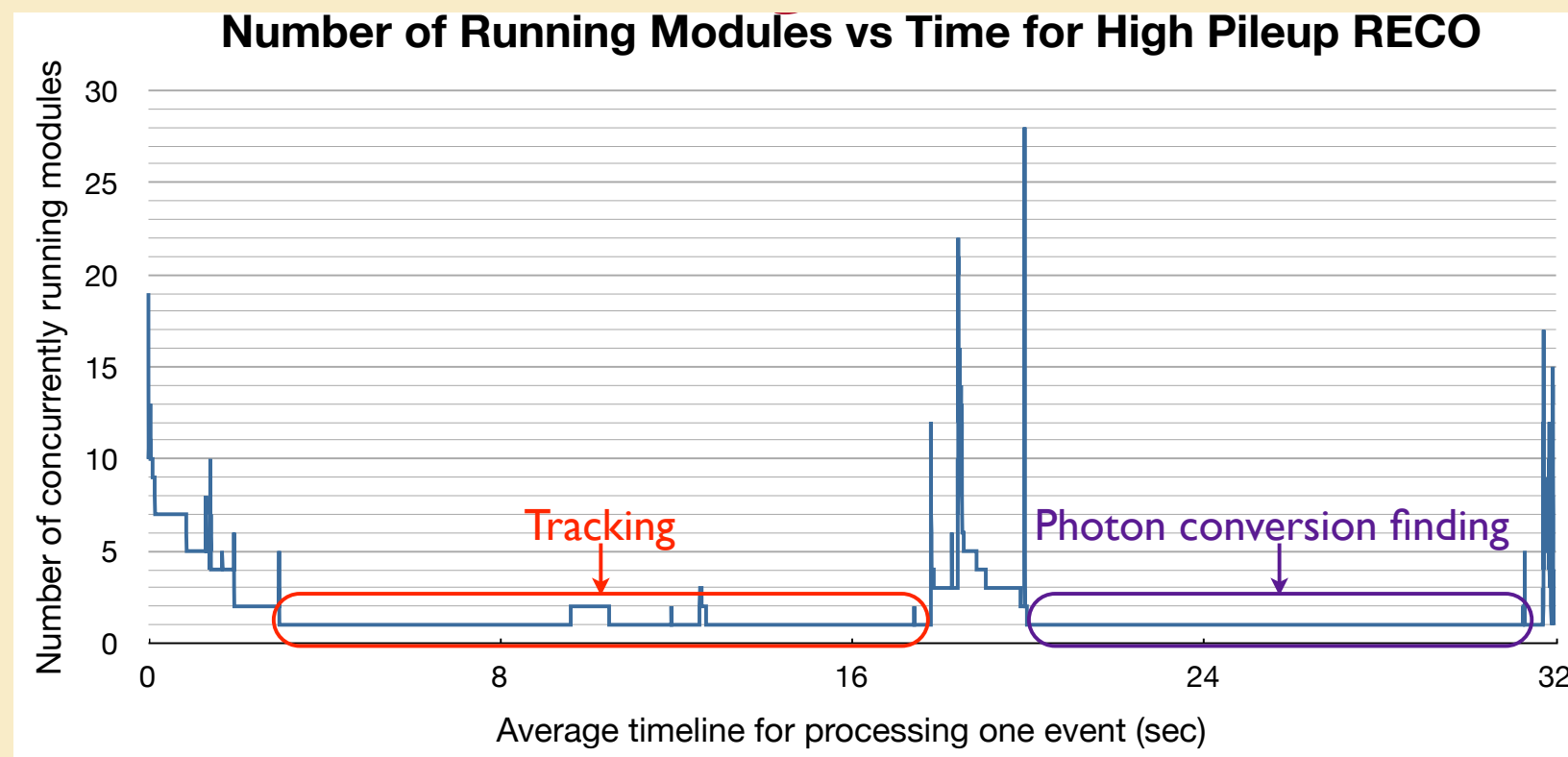
Parallel tasks with dependencies

There are threading frameworks that help deal with dependencies

Intel Thread Building blocks (TBB)

Apple libdispatch

Example: CMS reconstruction. Deep parallelization may be necessary

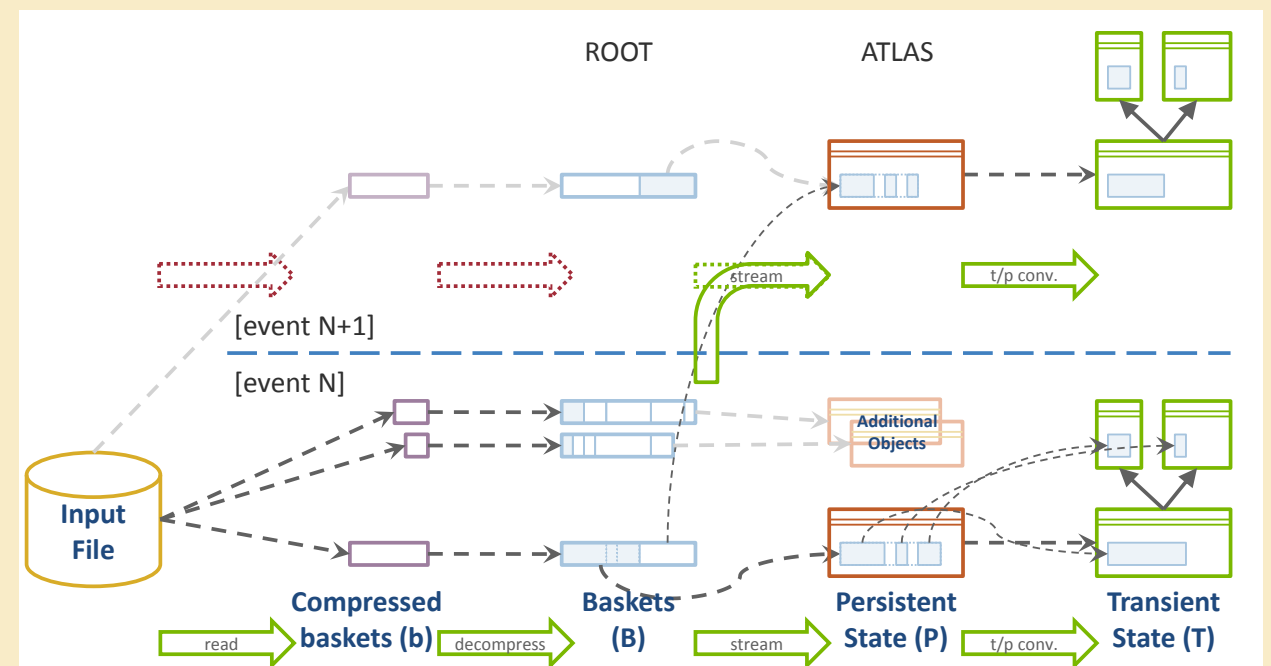
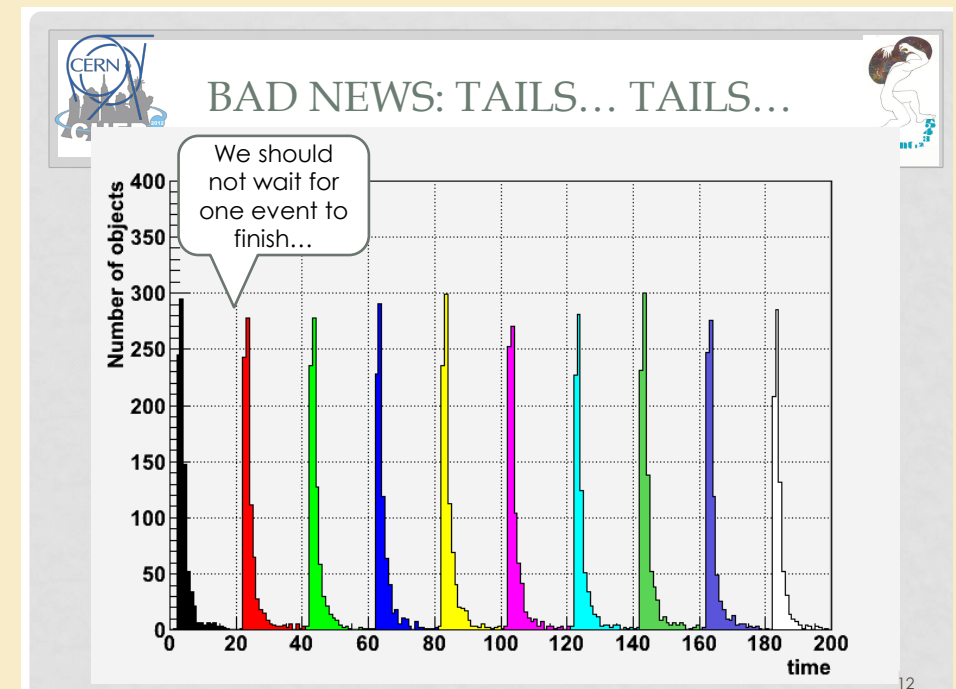


New ideas, new problems

Perhaps blur the boundary between events
(e.g. have a core doing particle transport
continuously)

Another difficulty is machines with
lots of cores will need lots of
i/o and network bandwidth

If multiple cores are reading data,
may repeat decompression, etc



Art and multicore processing

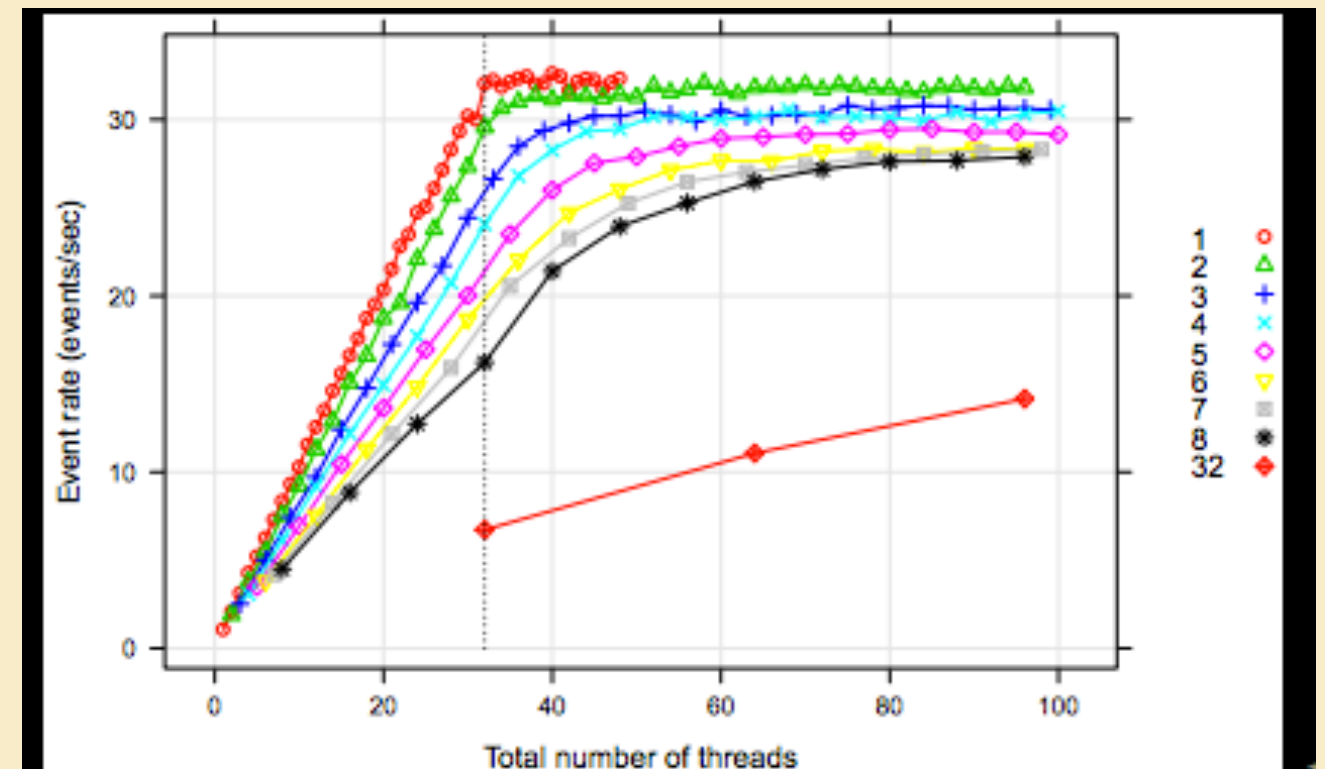
Ongoing R&D effort

Geant4 and Root have concurrency efforts too

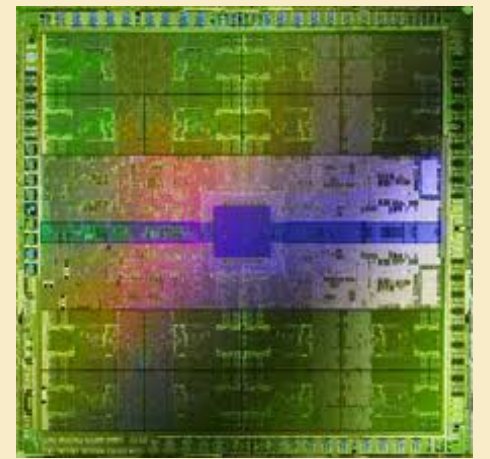
Art's path structure lends itself to multicore processing (each path is run by a core, Art knows the path dependencies)

Currently studying with NOvA track finding and TBB

Useful for us? Probably yes for
Geant4 simulations



General Purpose GPUs



Massively parallel systems on a chip

Thousands of cores

GPUs are best suited for “Data Parallelism” - running the same task on different pieces of distributed data

Contrast with multicore CPUs - best suited for “Task Parallelism”

Can see orders of magnitude speed improvements over CPUs for appropriate use cases (e.g. 200x speed up)

Have to use C/C++ extensions (CUDA)

GPU Farms and “clouds” exist – challenge is to integrate with CPU based workflows

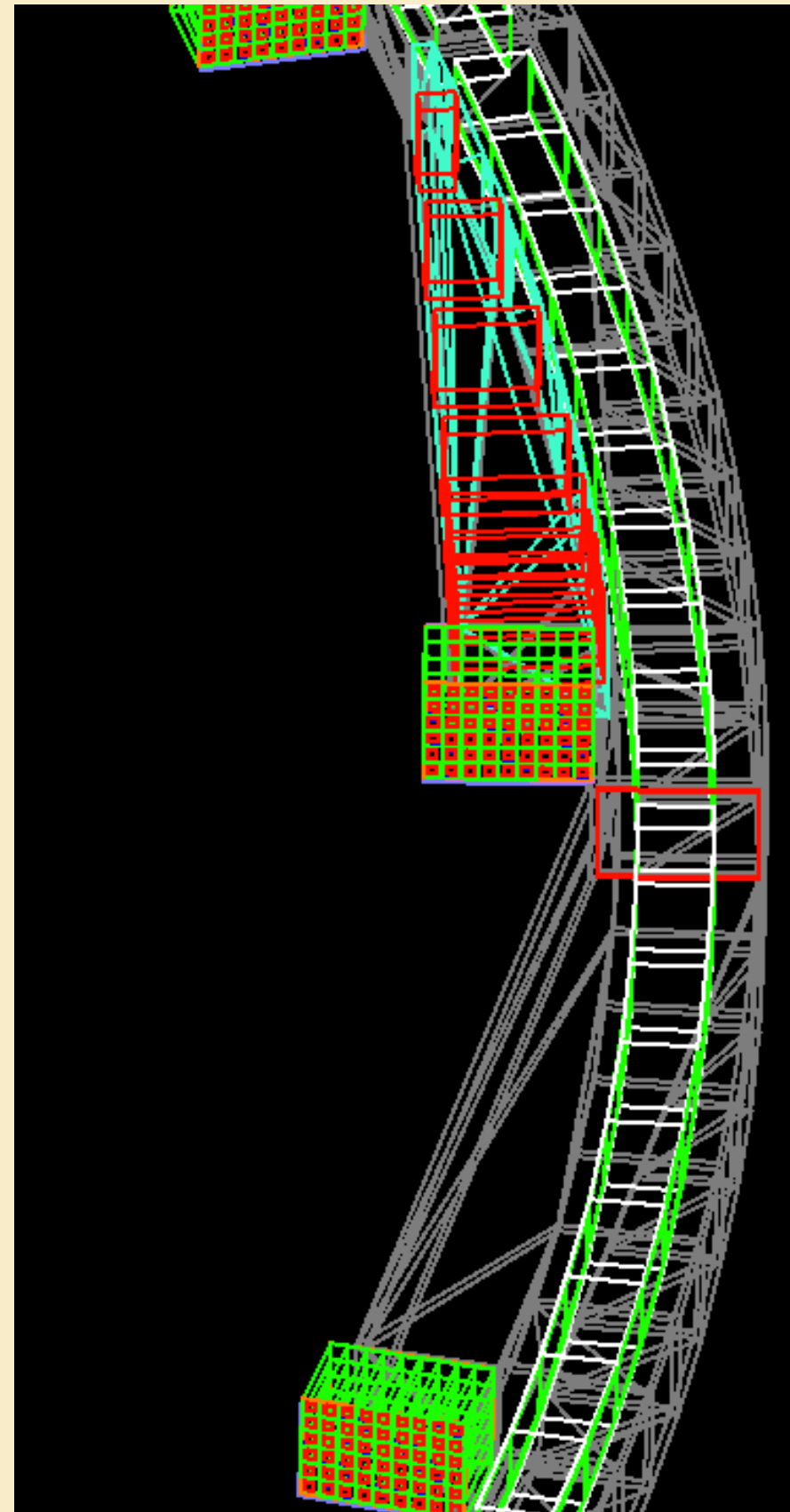
Fermilab recently installed a 152 GPU farm for Lattice QCD

Muon g-2 online makes use of a GPU farm - offline is not a good use case yet

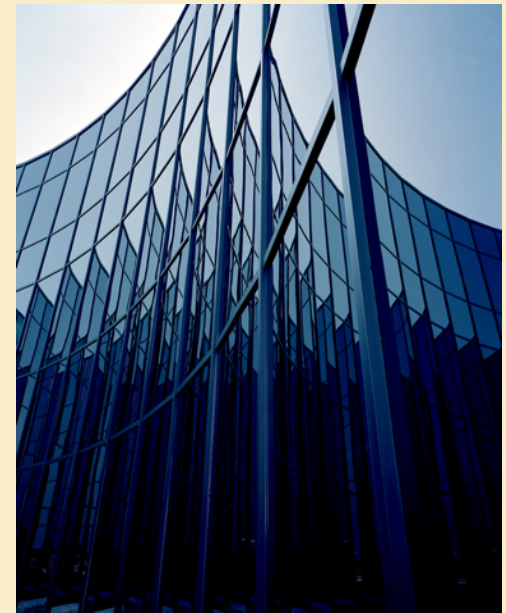
Back to g-2 -- where do we stand with computing?

We are outfitted with interactive VMs, batch slots, tape, and disk similar to the other IF experiments. We will soon have SAM cache to complete the picture.

The main Art activity is porting g2migtrace. The main coding is done. The big effort to do the port was very successful, but now we need a good analysis example to keep the momentum going (validation and testing) and attract users.



Summary



Fermilab has a vision of computing involving efficient use of a common infrastructure and common tools

Fermilab computing is seen more as a service the lab provides to experiments as opposed to experiments driving individual resources

This is perfect for us: let's concentrate on the physics and not the computing

Much of the technology and improvements we just use (probably without even knowing it). But realize there's lots of work going on behind the scenes

Muon g-2 is well situated to take advantage of the computing the lab offers